IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of	:	
Takehiro YOSHIDA et al.	:	
Serial No. NEW	:	**Attn: APPLICATION BRANCH**
Filed April 1, 2004	:	Attorney Docket No. 2004-0504A

PROGRAM LINKING PROGRAM, PROGRAM PRODUCT, PROGRAM LINKING DEVICE, TERMINAL DEVICE, AND PROGRAM LINKING METHOD

THE COMMISSIONER IS AUTHORIZED TO CHARGE ANY DEFICIENCY IN THE FEES FOR THIS PAPER TO DEPOSIT ACCOUNT NO. 23-0975

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2003-097892, filed April 1, 2003, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Takehiro YOSHIDA et al.

By _____
Jeffrey R. Filipek
Registration No. 41,471
Attorney for Applicants

JRF/nk
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
April 1, 2004

P13415

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００３年　４月　１日

出願番号
Application Number: 特願２００３－０９７８９２

[ST. 10/C]: ［ＪＰ２００３－０９７８９２］

出願人
Applicant(s): 松下電器産業株式会社

２００３年１２月２６日

特許庁長官
Commissioner,
Japan Patent Office 今井康

出証番号　出証特２００３－３１０８０３４

【書類名】　特許願
【整理番号】　2968150023
【提出日】　平成15年 4月 1日
【あて先】　特許庁長官殿
【国際特許分類】　G06F 9/445
G06F 13/00 354
【発明者】
【住所又は居所】　愛知県名古屋市中区栄２丁目６番１号白川ビル別館５階
株式会社松下電器情報システム名古屋研究所内
【氏名】　吉田　健宏
【発明者】
【住所又は居所】　愛知県名古屋市中区栄２丁目６番１号白川ビル別館５階
株式会社松下電器情報システム名古屋研究所内
【氏名】　川本　琢二
【発明者】
【住所又は居所】　愛知県名古屋市中区栄２丁目６番１号白川ビル別館５階
株式会社松下電器情報システム名古屋研究所内
【氏名】　河合　正樹
【特許出願人】
【識別番号】　000005821
【氏名又は名称】　松下電器産業株式会社
【代理人】
【識別番号】　100097445
【弁理士】
【氏名又は名称】　岩橋　文雄
【選任した代理人】
【識別番号】　100103355
【弁理士】
【氏名又は名称】　坂口　智康

【選任した代理人】

【識別番号】　100109667

【弁理士】

【氏名又は名称】　内藤　浩樹

【手数料の表示】

【予納台帳番号】　011305

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9809938

【書類名】　明細書

【発明の名称】　プログラムリンクプログラム、プログラムリンクプログラムを記録した記録媒体、プログラムリンク装置及びプログラムリンク方法

【特許請求の範囲】

【請求項１】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記記憶手段の使用状況の判断に応じて、前記リンク後プログラムを作成するために必要なリンクの一部だけを行うか或いは全く行わないことがある、

プログラムリンクプログラム。

【請求項２】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で優先的にリンクを行う前記リンク前プログラムを決定するために、各リンク前プログラムがリンク後プログラム作成のために使われている頻度、或いは、各リンク前プログラムのサイズ、或いは、各リンク前プログラムがリンク後プログラム作成のために使われている頻度とそのリンク前プログラムサイズとの積、或いは、各リンク前プログラムの実行時リンク時間、或いは、各リンク前プログラムの実行頻度、の何れか１又は２以上を使用する、

プログラムリンクプログラム。

【請求項３】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使

用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムがリンク後プログラム作成のために使われている頻度が低いものから優先的にリンクを行う、

プログラムリンクプログラム。

【請求項４】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムのサイズが小さなものから優先的にリンクを行う、

プログラムリンクプログラム。

【請求項５】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムを実行時にリンクするために必要な時間が大きなものから優先的にリンクを行う、

プログラムリンクプログラム。

【請求項６】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度

で前記リンク後プログラムを作成するために、各リンク前プログラムが実行される頻度の高いものから優先的にリンクを行う、

プログラムリンクプログラム。

【請求項７】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムがリンク後プログラム作成のために使われている頻度とそのリンク前プログラムサイズとの積が小さなものから優先的にリンクを行う、

プログラムリンクプログラム。

【請求項８】　コンピュータ読み取り可能なデータ記録媒体であって、

請求項１から請求項７の何れか１項に記載のプログラムリンクプログラムを記録した記録媒体。

【請求項９】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、

前記リンク後プログラムを記憶手段に記憶するステップと、

前記記憶手段の使用状況を判断するステップとを有し、

前記記憶手段の使用状況の判断に応じて、前記リンク後プログラムを作成するために必要なリンクの一部だけを行うか或いは全く行わないことがある、

プログラムリンク方法。

【請求項１０】　複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成する手段と、

前記リンク後プログラムを記憶手段に記憶する手段と、

前記記憶手段の使用状況を判断する手段とを有し、

リンク後プログラムを作成する手段は、前記記憶手段の使用状況の判断に応じて、前記リンク後プログラムを作成するために必要なリンクの一部だけを行うか

或いは全く行わないことがある、

プログラムリンク装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本特許出願に係る発明は、いわゆるソースプログラム即ち原始言語で記述されたプログラムをコンパイラによって変換して得られる中間言語プログラム、具体的には例えば実行時の絶対アドレスが決定されていないプログラムや汎用のライブラリプログラムを複数結合し、例えば実行時の絶対アドレスが決定されたオブジェクトプログラムを作成するリンカに関するものである。

【０００２】

しかし上記、実行時の絶対アドレスの決定の有無は１つの例示であってこれに限定されるものではない。

【０００３】

特に、実行時環境でのメモリサイズに制限の大きい、携帯機器を始めとする各種装置で実行される、例えばＪａｖａ（Ｒ）言語で記述されたプログラムのリンカに関するものである。

【０００４】

しかしこれらもあくまで１つの例示であって、これに限定されるものではない。

【０００５】

【従来の技術】

従来の、Ｊａｖａ（Ｒ）言語やＣ言語で記述されたプログラムを複数結合し、実行プログラムを作成するリンカには、例えば、下記特許文献１、特許文献２に記載のものがあった。

【０００６】

例えば従来のＪａｖａ（Ｒ）実行環境では、アプリケーションプログラムのＪａｖａ（Ｒ）ソースコードを中間言語のバイトコードに変換し、各クラス単位でファイルとして記憶される。

【０００７】

アプリケーションが起動されると、これらのクラスファイルから必要なものをロードし、これらをリンクして実行プログラムが作成される。

【０００８】

この方式を実行時リンク方式と呼び、この時のプログラムと処理の流れを図９に示す。

【０００９】

図９に示す従来の実行時リンク方式のリンカでは、中間言語で記述されているクラスＡ、クラスＢ、クラスＸ、クラスＹの各プログラムが例えばサーバ９０９に記憶されている。

【００１０】

これらの４つのプログラムは事前に端末装置９０１にロードされ記憶される。

【００１１】

例えばこれらの中からクラスＡ、クラスＢ、クラスＸの各プログラムをリンクすることによってアプリケーションプログラム９０２が作成されると仮定する。

【００１２】

このアプリケーションプログラム９０２が端末装置９０１で起動されると、その時点で実行時リンクが行われ、クラスＡ、クラスＢ、クラスＸの各プログラムをリンクすることによってアプリケーションプログラム９０２が作成され、実行される。

【００１３】

例えばこれらの中からクラスＡ、クラスＢ、クラスＹの各プログラムをリンクすることによってアプリケーションプログラム９０３が作成されると仮定する。

【００１４】

このアプリケーションプログラム９０３が端末装置９０１で起動されると、その時点で実行時リンクが行われ、クラスＡ、クラスＢ、クラスＹの各プログラムをリンクすることによってアプリケーションプログラム９０３が作成され、実行される。

【００１５】

又、例えば従来のＣ言語実行環境では、ソースプログラムをコンパイルしてオブジェクトプログラムに変換し、同時に必要な全てのライブラリを事前にリンクして実行形式ファイルを作成して記憶する。

【００１６】

アプリケーションが起動されると、この実行形式ファイルがロードされ、実行される。

【００１７】

この方式を事前リンク方式と呼び、この時のプログラムと処理の流れを図１０に示す。

【００１８】

図１０に示す従来の事前リンク方式のリンカでは、中間言語で記述されているクラスＡ、クラスＢ、クラスＸ、クラスＹの各プログラムが例えばサーバ９２９に記憶されていると同時に、サーバ９２９で事前リンクが行われ、アプリケーションプログラム９２２とアプリケーションプログラム９２３が作成されている。

【００１９】

端末装置９２１はこれらのアプリケーションプログラム９２２とアプリケーションプログラム９２３をロードして記憶する。

【００２０】

そしてこれらのアプリケーションプログラム９２２とアプリケーションプログラム９２３が起動されると、即時に実行することができる。

【００２１】

【特許文献１】

特開平１０－０６９３７６号公報

【特許文献２】

特表２０００－５１４５８４号公報

【００２２】

【発明が解決しようとする課題】

しかしながら例えば前記図９に示す実行時リンク方式では、端末装置９０１がロードして記憶するのは、クラスＡ、クラスＢ、クラスＸ、クラスＹの各中間言

語プログラムを１つずつであるから、重複してライブラリを記憶する必要が無く、記憶領域を節減することができる。

【００２３】

これは例えば携帯機器のように記憶部の容量に制限があるような機器では大きなメリットではあった。

【００２４】

しかし一方、アプリケーションが起動されてからリンクを行うため、アプリケーションの起動から実際に実行されるまで長い時間を要すると言う欠点があった。

【００２５】

又、例えば前記図１０に示す事前リンク方式では、先にリンクを行って実行形式ファイルを作っておくため、アプリケーションが起動されるとすぐに実行することが可能であり、このことは大きなメリットである。

【００２６】

しかし他方、事前にリンクを行っておくことによって、端末装置９２１は、アプリケーションプログラム９２２とアプリケーションプログラム９２３とで共通に使用するライブラリであるクラスＡとクラスＢを、重複して記憶しなければならない。

【００２７】

これは例えば前記携帯機器のように記憶部の容量に制限があるような機器では大きな欠点であった。

【００２８】

【課題を解決するための手段】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記記憶手段の使用状況の判断に応じて、前記リンク後プログラムを作成するために必要なリンクの一部だけを行うか或いは全く行わないことがある、プログラムリンクプログラムである。

【００２９】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で優先的にリンクを行う前記リンク前プログラムを決定するために、各リンク前プログラムがリンク後プログラム作成のために使われている頻度、或いは、各リンク前プログラムのサイズ、或いは、各リンク前プログラムがリンク後プログラム作成のために使われている頻度とそのリンク前プログラムサイズとの積、或いは、各リンク前プログラムの実行時リンク時間、或いは、各リンク前プログラムの実行頻度、の何れか１又は２以上を使用する、プログラムリンクプログラムである。

【００３０】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムがリンク後プログラム作成のために使われている頻度が低いものから優先的にリンクを行う、プログラムリンクプログラムである。

【００３１】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムのサイズが

小さなものから優先的にリンクを行う、プログラムリンクプログラムである。

【００３２】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムを実行時にリンクするために必要な時間が大きなものから優先的にリンクを行う、プログラムリンクプログラムである。

【００３３】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムが実行される頻度の高いものから優先的にリンクを行う、プログラムリンクプログラムである。

【００３４】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記リンク後プログラムを作成するステップの実行によって前記記憶手段の使用状況が前記記憶手段の記憶容量を超える時には、前記記憶容量を超えない限度で前記リンク後プログラムを作成するために、各リンク前プログラムがリンク後プログラム作成のために使われている頻度とそのリンク前プログラムサイズとの積が小さなものから優先的にリンクを行う、プログラムリンクプログラムである。

【００３５】

本特許出願の発明は、コンピュータ読み取り可能なデータ記録媒体であって、請求項１から請求項７の何れか１項に記載のプログラムリンクプログラムを記録した記録媒体である。

【００３６】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成するステップと、前記リンク後プログラムを記憶手段に記憶するステップと、前記記憶手段の使用状況を判断するステップとを有し、前記記憶手段の使用状況の判断に応じて、前記リンク後プログラムを作成するために必要なリンクの一部だけを行うか或いは全く行わないことがある、プログラムリンク方法である。

【００３７】

本特許出願の発明は、複数のリンク前プログラムから１又は複数個をリンクしてリンク後プログラムを作成する手段と、前記リンク後プログラムを記憶手段に記憶する手段と、前記記憶手段の使用状況を判断する手段とを有し、リンク後プログラムを作成する手段は、前記記憶手段の使用状況の判断に応じて、前記リンク後プログラムを作成するために必要なリンクの一部だけを行うか或いは全く行わないことがある、プログラムリンク装置である。

【００３８】

【発明の実施の形態】

本特許出願に係る発明（以後、本発明とも言う）の実施の形態を図面を使って詳細に説明する。

【００３９】

（実施の形態１）

本特許出願に係る発明（省略して「本発明」とも言う）であるプログラムリンクプログラム又はプログラムリンク方法（省略して単に「リンク」とも言う）を実行するプログラムリンク装置の一実施の形態である端末装置１０１のブロック構成図を図１に示す。

【００４０】

この端末装置１０１とは更に具体的には、携帯電話や携帯情報端末（ＰＤＡ：Ｐｅｒｓｏｎａｌ　Ｄｉｇｉｔａｌ　Ａｓｓｉｓｔａｎｔｓ）等の携帯用情報機器、或いはパソコン等の情報機器、又はマイコンによって制御される種々の家庭用電気製品、事務用電子機器、その他制御装置等である。

【００４１】

更に、他の機器、例えば図１のサーバ１０９から中間言語プログラムをダウンロードして実行する装置である。

【００４２】

但しこのサーバ１０９は１つの例であって、必ずしもサーバ１０９に限るものではなく、他の種々の電子機器、例えば通常のパソコンや電気製品であっても良い。

【００４３】

この端末装置１０１で主にリンクを実行するのは事前リンク部１４１である。

【００４４】

この事前リンク部１４１の作用を中心に、図１、図２、図３、図４、図５を使って説明する。

【００４５】

この端末装置１０１では様々なアプリケーションが実行される。例えば図４、図５に示すアプリケーション１、アプリケーション２、アプリケーション３が実行され、これを実際に実行するのはアプリケーションプログラム１、アプリケーションプログラム２、アプリケーションプログラム３である。

【００４６】

特に区別する必要が無い時には、アプリケーションプログラム１、アプリケーションプログラム２、アプリケーションプログラム３と、アプリケーション１、アプリケーション２、アプリケーション３との両方の意味を含めて、単にアプリケーション１、アプリケーション２、アプリケーション３とも言う。

【００４７】

又同様に、特に区別する必要が無い時には、アプリケーションプログラムとアプリケーションとの両方の意味を含めて、単にアプリケーションとも言う。

【００４８】

これらのアプリケーションプログラムは、幾つかの中間言語で記述されたクラスライブラリから構成されており、この中間言語のクラスライブラリはサーバ１０９に記憶されている。

【００４９】

図１のサーバ１０９にはクラスＡ、クラスＢ、クラスＸ、クラスＹの４つのクラスだけが記載されているが、勿論これらだけに限るものではなく、その他必要となる全てのクラスライブラリが記憶されているが、図１では記載を省略している。

【００５０】

端末装置１０１の制御部１３１でダウンロード命令１１３が実行されると、サーバ１０９から端末装置１０１にこれらのクラスライブラリがダウンロードされる。

【００５１】

このダウンロードは図示していないが、種々のネットワーク、例えばインターネットやＬＡＮ、ＣＡＴＶ、衛星回線、電話回線、ＩＳＤＮ、ＡＴＭ、その他のネットワークを経由してダウンロードされることもあり、又一方いわゆるパッケージ形態、即ち何等かのパッケージメディアに記録された状態で端末装置１０１に読み込まれることもある。

【００５２】

又、ダウンロードの指定方法には、端末装置１０１が実行するアプリケーションの中の特定のアプリケーションが指定され、その指定されたアプリケーションの実行のために必要なクラスライブラリだけをダウンロードする方法と、端末装置１０１が実行し得る全てのアプリケーションの実行のために必要なクラスライブラリをダウンロードする方法と、サーバ１０９が保有している全てのクラスライブラリをダウンロードする方法と、その他、種々の方法でクラスライブラリが指定されその指定されたクラスライブラリがダウンロードされる方法、その他がある。

【００５３】

このダウンロードを行うのはダウンロード制御部１３３であり、ダウンロードされたクラスライブラリはファイルシステム部１３７に記憶保存される。

【００５４】

制御部１３１で事前リンク命令１１５が実行されると、ファイルシステム部１３７に記憶保存されているダウンロードされたクラスライブラリの事前リンクが行われる。

【００５５】

前記の通り、各アプリケーションの実行に先立って事前にリンクしておくことにより、実行時の速度を向上するためである。

【００５６】

又、この事前リンク命令の実行は、ダウンロードのタイミングに同期して、或いはダウンロードのタイミングと同時に行われても良いし、ある所定の時間によって行われても良いし、ユーザの何等かの操作に対応して行われても良いし、その他の何等かのタイミングによって行われても良い。

【００５７】

事前リンク命令１１５が実行されると制御部１３１から事前リンク部１４１に対して事前リンク実行の制御が行われ、以下の処理がスタートする（図２、２２０）。

【００５８】

事前リンク部１４１では、まず最初に全てのクラスライブラリのリンクが行われる（図２、２２１）。

【００５９】

これはクラス呼び出し関係検出部１４６が、各アプリケーション毎に呼び出しているクラスライブラリを検出し、更に呼び出されているクラスライブラリがまた呼び出しているクラスライブラリを検出し、この処理を繰り返すことによって行われる。

【００６０】

呼び出されていることが検出されたクラスライブラリは結合部１４３によって結合（リンク）される。この結合（リンク）は実行時アドレスの決定を伴うこと

もあるし、必ずしも実行時アドレスの最終的な決定まで到らないこともあり、どちらかに限るものではない。

【００６１】

このリンク処理を実行している時、メモリ量管理部１４４は各リンクの実行によってファイルシステム部１３７のメモリ量（残り容量）がどのように変化するかを管理している。

【００６２】

以上の処理を何回か必要な回数繰り返すことによって、端末装置１０１が実行できる、或いは実行しなければならない、或いは指定された、全てのアプリケーションプログラムのリンクが終了する（図２、２２３）。

【００６３】

この時点で、メモリ量管理部１４４は、管理しているファイルシステム部１３７のメモリ量（残り容量）を確認し、その容量内に収まっているか、或いはメモリオーバーフローしているかを判断する（図２、２２５）。

【００６４】

もしもファイルシステム部１３７のメモリ容量内に収まっているならば、全てのアプリケーションプログラムが正常にリンクできたことになり、事前リンクは終了する（図２、２２８）。

【００６５】

もしもファイルシステム部１３７のメモリ容量をオーバーフローしているならば、リンク数検出部１４２が、現在リンクに使われているクラスライブラリの中で、最も多くのアプリケーションのために使用されているクラスライブラリを検出し、そのクラスライブラリを使用しているアプリケーション数（最大値に相当）を求める（図２、２３１）。説明の都合上、その数を仮にＮとする（図２、２３１）。

【００６６】

そして当初は「全てのクラスライブラリのリンクを行う」と設定されていたのが、これ以降は「Ｎ個以上のアプリケーションで使用されているクラスライブラリのリンクは行わない」と、設定の変更が行われる（図２、２３３）。

【００６７】

そして一旦既に行った全てのアプリケーションプログラムのリンクを解除し、再び新しい設定に従って全てのアプリケーションのリンクを行う（図２、２２３）。

【００６８】

尚ここで、一旦既に行った全てのアプリケーションのリンクを解除する方法として、リンク済みのプログラムのリンクを分解する方法や、リンク前のプログラムを別に保存しておく方法、その他の方法が有り得るが、どのような方法を使用しても構わない。

【００６９】

例えばリンク前のクラスライブラリをダウンロード制御部１３３に保存しておく方法も有り得るし、例えば再度ダウンロード制御部１３３がサーバ１０９からリンク前のクラスライブラリをダウンロードする方法も有り得る。

【００７０】

或いは結合ファイル解除部１３５がリンク済みのプログラムのリンクを分解し、再度ファイルシステム部１３７に渡す方法も有り得る。これらの何れの方法を使用しても構わない。

【００７１】

例えば前記の通り、この端末装置１０１で実行されるアプリケーションが、例えば図４、図５に示すアプリケーション１、アプリケーション２、アプリケーション３であったと仮定する。

【００７２】

このケースでは、全アプリケーションの実行に必要となるクラスライブラリは、Ｘ、Ｙ、Ａ、Ｂ、Ｃ、Ｄ、Ｚであり（省略してクラスライブラリを特定する記号だけを記している、以後同じ）、これらがサーバ１０９からダウンロードされ記憶保存されている。

【００７３】

全アプリケーションプログラムのリンクが実行されることによって（図２、２２３）、図４に示すステップ１、ステップ２、ステップ３が実行され、端末装置

１０１に全クラスライブラリがリンクされたアプリケーションプログラム１、アプリケーションプログラム２、アプリケーションプログラム３が作成される（図４、図５、ステップ３）。

【００７４】

この状態では、全てのクラスライブラリが既にリンクされているので、図１の実行時リンク部１５２が実行時に改めてリンクを行うことは必要でなく、実行時のリンク数は０である（図５、ステップ３）。

【００７５】

しかし一方、重複を含めずに数えた全クラスライブラリ数は上記７であるが、重複を含めて数えた全クラスライブラリ数は１１であり、４つのクラスライブラリが重複して記憶されている（図５、ステップ３）。この重複する４つのクラスライブラリとは２つのＤと、１つずつのＢとＣである。

【００７６】

この重複する４つのクラスライブラリの分だけ、ファイルシステム部１３７のメモリ容量をオーバーフローする可能性が高くなっている。

【００７７】

そこでこれらクラスライブラリの中で、最も多くのアプリケーションプログラムで使用されているものが、リンク数検出部１４２によって検出される（図２、２３１）。

【００７８】

今の例では、このクラスライブラリはＤであり、３つのアプリケーション全てで使用されている（図５、ステップ４）。従って、上記Ｎは３になる。

【００７９】

そこで以降は、「３個以上のアプリケーションで使用されているクラスライブラリはリンクしない」と設定し直され（図２、２３３）、改めて全てのリンクが行われる（図２、２２３）。

【００８０】

尚、図４のステップ４では、クラスライブラリＤのリンクを解除するように記載している通り、必ずしも一旦全てのクラスライブラリのリンクを解除してリン

クし直す方法に限るものではなく、例えば３個以上のアプリケーションで使用されているクラスライブラリだけを検出しながらそのリンクを解除する方法でも構わない。

【００８１】

又、図２に示す上記の方法では一旦全てのアプリケーションのリンクを実行してからメモリ容量のオーバーフローを検出しているが、図３には１つのアプリケーションのリンクを行う毎にメモリ容量のオーバーフローを検出する方法を示している。どちらの方法も本質的な部分は同一であるから、後者の方法については詳細な説明を省略する。

【００８２】

このようにクラスライブラリＤのリンクを解除した状態では、アプリケーションプログラム１～３の全てで実行時にクラスライブラリＤのリンクを実行することが必要になるので、必要な実行時リンク数は３である（図５、ステップ４）。

【００８３】

そのためにある程度実行時の速度は低下するが、その一方、重複を含めて数えた全クラスライブラリ数は９であり（図５、ステップ４）、重複して記憶されているクラスライブラリ数は２に減少している。その分、ファイルシステム部１３７がメモリ容量をオーバーフローする可能性は低くなる。この重複する２つのクラスライブラリとは、１つずつのＢとＣである。

【００８４】

従って、図２の２２５では、今回の２周目は前回よりもメモリオーバーフローする可能性は低くなる。

【００８５】

そこでメモリオーバーフローが発生しなければ、終了する（図２、２２８）。

【００８６】

しかしこれでもファイルシステム部１３７がメモリ容量をオーバーフローしたと仮定する（図２、２２５）。このケースは今の例では、図２の２２５が２周目の判断でもメモリオーバーフローを検出した時に相当する。

【００８７】

このケースでは再び図２の２３１で、現在リンクされているクラスライブラリの中で、最も多くのアプリケーションによって使用されているものが、リンク数検出部１４２によって検出される。

【００８８】

それは図５のステップ５に示す通り、ＢとＣであり、これらは共に２つのアプリケーションで使用されている。

【００８９】

従って次に、「２個以上のアプリケーションで使用されているクラスライブラリはリンクしない」と再設定される（図２、２３３、２周目）。

【００９０】

そして再び一旦全てのリンクが解除された後、図２の２２３で３度目のリンクが実行される。

【００９１】

念のため説明すると、１周目のリンクでは全てのクラスライブラリのリンクが行われ、２周目のリンクでは３個以上のアプリケーションで使用されているクラスライブラリのリンクが行われず、今回の３周目では２個以上のアプリケーションで使用されているクラスライブラリのリンクが行われない。

【００９２】

或いは、２個以上のアプリケーションで使用されているクラスライブラリのリンクが解除されるとしても良い。

【００９３】

この状態を図５のステップ５に示す。

【００９４】

この状態では、クラスライブラリＤ、Ｂ、Ｃのリンクが行われていないので、アプリケーションの実行時にはこれらのクラスライブラリから必要なクラスライブラリのリンクを行うことが必要であり、その総数は７である。

【００９５】

何故ならば、アプリケーション１の実行時には、Ｘのクラスライブラリを基準に考えると、このＸに対してＢ、Ｃ、Ｄの３つをリンクする必要があり、アプリ

ケーション２の実行時には、ＹとＡのクラスライブラリは既にリンクされているが、このリンクされたものに対してＢ、Ｄの２つをリンクする必要があり、アプリケーション３の実行時には、Ｚのクラスライブラリを基準に考えると、このＺに対してＣ、Ｄの２つをリンクする必要があるからである。

【００９６】

従って、実行時の速度はその分低下するが、少なくともアプリケーション２で必要なクラスライブラリＹとＡとが既にリンクされているだけ何も事前リンクされていないよりも実行時の速度は向上する。

【００９７】

一方、重複して記憶されているクラスライブラリは存在しない。従ってメモリ容量の無駄はその意味では全く生じていない。

【００９８】

このことは、前記「２個以上のアプリケーションで使用されているクラスライブラリはリンクしない」と言う条件は、「唯１つのアプリケーションで使用されているクラスライブラリしかリンクしない」ことを意味し、この条件では明らかにクラスライブラリを重複して記憶することは生じ得ないことからも理解することができる。

【００９９】

図１の端末装置１０１の制御部において、実行命令１１７が実行されると制御部１３１から実行制御部１５１に対してアプリケーションの実行が指示される。

【０１００】

この実行命令１１７の実行はユーザの何等かの操作や、指示や、他の機器からの制御や、時間その他の起動による場合等がある。

【０１０１】

又、通常このアプリケーションの起動はどのアプリケーションを起動するのか、例えば今までの例では、アプリケーション１か、アプリケーション２か、アプリケーション３なのかを指定して行われる。

【０１０２】

実行制御部１５１に特定の指定されたアプリケーションの実行が指示されると

、実行制御部１５１の実行時リンク部１５２がそのアプリケーションを実行するためにまだ幾らかのクラスライブラリのリンクすることが必要であるか否かを判断し、もしも必要であるならばそのクラスライブラリのリンクを実行する。

【０１０３】

例えば、図５のステップ３の状態であるならば、どのアプリケーションの実行が指示されたとしても、実行時にリンクしなければならないクラスライブラリは存在しない。

【０１０４】

例えば、図５のステップ４の状態で、アプリケーション１～３の何れかの実行が指示されたとすると、クラスライブラリＤの実行時リンクが必要になり、実行時リンク部１５２はこのクラスライブラリＤのリンクを実行する。

【０１０５】

例えば、図５のステップ５の状態で、アプリケーション１の実行が指示されたとすると、クラスライブラリＢとＣとＤの実行時リンクが必要となり、実行時リンク部１５２はこのクラスライブラリＢとＣとＤのリンクを実行する。

【０１０６】

例えば、図５のステップ５の状態で、アプリケーション２の実行が指示されたとすると、クラスライブラリＢとＤの実行時リンクが必要となり、実行時リンク部１５２はこのクラスライブラリＢとＤのリンクを実行する。

【０１０７】

例えば、図５のステップ５の状態で、アプリケーション３の実行が指示されたとすると、クラスライブラリＣとＤの実行時リンクが必要となり、実行時リンク部１５２はこのクラスライブラリＣとＤのリンクを実行する。

【０１０８】

このようにして実行に必要な全てのクラスライブラリのリンクが行われると、実行部１５３がそのアプリケーションプログラムの実行を行う。

【０１０９】

（実施の形態２）

次に本発明の第２の実施の形態であるプログラムリンクプログラム、プログラ

ムリンク方法、プログラムリンク装置について説明する。但し、前記第１の実施の形態と本質的に異なる部分についてのみ説明し、前記第１の実施の形態と本質的に同一部分については説明を省略する。

【０１１０】

この第２の実施の形態でも、全てのアプリケーションプログラムのリンクを行って、ファイルシステム部１３７のメモリ容量内に収まっているならば、事前リンクは正常に終了する。

【０１１１】

このことに関して、本発明の第１の実施の形態と同じである。

【０１１２】

もしもファイルシステム部１３７のメモリ容量をオーバーフローするならば、クラスライブラリの中で、より少ないアプリケーションのためにリンクされているクラスライブラリから優先的にリンクを行う。このことに関しても、本発明の第１の実施の形態と同じである。

【０１１３】

この第２の実施の形態が前記第１の実施の形態と本質的に異なる部分は、より少ないアプリケーションのためにリンクされているクラスライブラリから優先的にリンクを行うための手順に関してのみである。

【０１１４】

従って、この手順についてのみ図６を参照して説明する。

【０１１５】

前記第１の実施の形態では、全てのリンクを実際に行うことによってファイルシステム部１３７のメモリ容量をオーバーフローするか否かを判定した。しかしメモリ容量の判定のために実施のリンクを行うことは非効率な部分を含む。

【０１１６】

そこでこの第２の実施の形態では、実際にリンクを行う前に、メモリサイズのチェックを行い、ファイルシステム部１３７のメモリに収容可能な最大限のライブラリを上記優先基準に基づいて予め求め、実際のリンクはそのライブラリに関してのみ、その後に実行する。

【０１１７】

図６はそのための手順を示している。

【０１１８】

この実施の形態では、まず、各ライブラリ毎に、そのライブラリがアプリケーションからリンクされる数、即ち、そのライブラリをリンクしているアプリケーションの数を「共有数」と定義し、この値を説明のため（ｎ）と記して、この（ｎ）を各ライブラリ毎に調べることによって、求める（図６、６２２）。

【０１１９】

次に、全ライブラリをこの共有数（ｎ）でソートし、（ｎ）の値が小さなものから順に並べ、０（ゼロ）から始まるインデックス番号を付し、テーブルを作成する（図６、６２３）。

【０１２０】

同じ（ｎ）の値となるライブラリの順序についてはどのように設定しても構わないし、或いは他の何等かの方法によって順序を設定しても構わない。例えば１つの方法として、同じ（ｎ）の値となるライブラリの順序については後述する本発明の第３の実施の形態から第６の実施の形態で説明する判断基準を使用することができる。

【０１２１】

次に、処理のために必要な２つの変数として「インデックス変数」と「リンク総和量」とを用意し、それぞれ０（ゼロ）に初期化する（図６、６２４）。

【０１２２】

ここからループの処理に入り、最初に、前記インデックス変数の値が、ライブラリの総数と等しいか否かを判断する（図６、６２５）。

【０１２３】

これはメモリをオーバーフローすることなく、全てのライブラリのリンクが完了したか否かを判断するためである。

【０１２４】

即ち、メモリをオーバーフローすることなく、全てのライブラリのリンクが完了すれば、この判定で、前記インデックス変数の値がライブラリの総数と等しく

なる。

【０１２５】

しかし最初の段階では、インデックス変数の値はゼロで、ライブラリ総数は１以上であるから等しくはならない。

【０１２６】

そこで前記インデックス番号の値がインデックス変数の値に等しいライブラリの増加量をリンク総和量に加える（図６、６２６）。

【０１２７】

ここで各ライブラリの増加量とは、図６には明記していないが、そのライブラリのメモリサイズとそのライブラリをリンクしているアプリケーション数マイナス１との積である（図７、７２２参照）。

【０１２８】

即ち、このライブラリの増加量とは、そのライブラリをリンクすることによって新たに必要となるメモリ容量（必要メモリの増加量）に相当する。

【０１２９】

この段階で、リンク総和量が、ファイルシステム部１３７のメモリ容量をオーバーフローするか否かを判定する（図６、６２７）。

【０１３０】

もしもこの段階で、リンク総和量がファイルシステム部１３７のメモリ容量をオーバーフローするならば、現在のインデックス変数に相当するインデックス番号のライブラリをリンクするとメモリオーバーフローすることになるので、その１つ手前、即ちインデックス番号が現在のインデックス変数の値マイナス１のライブラリまでのリンクを実行する（図６、６２９）。

【０１３１】

もしもこの段階で、リンク総和量がファイルシステム部１３７のメモリ容量をオーバーフローしないならば、現在のインデックス変数に相当するインデックス番号のライブラリをリンクしてもメモリオーバーフローを生じないことが解る。

【０１３２】

そこで次にインデックス変数を１増加し（図６、６２８）、これによって全て

のライブラリに関してリンク可否の確認が終了するかの判断に戻る（図６、６２５）。これは前記処理ループの先頭である。

【０１３３】

これによってインデックス変数値が全ライブラリ数と等しくなれば、全てのライブラリの確認が終り、全ライブラリがリンク可能であることになり、全ライブラリのリンクを実行することができる（図６、６２９）。

【０１３４】

全ライブラリの確認が終わっていなければ、次のライブラリの増加量をリンク総和量に加え（図６、６２６）、リンク可能か否かの判断を繰り返す（図６、６２７）。

【０１３５】

この第２の実施の形態が前記第１の実施の形態と本質的に異なる部分は、以上説明した通り、リンクを実際に実行する前に、リンク可能なライブラリの範囲を確定することとそのための手順だけであるので、他に関しては説明を省略する。

【０１３６】

（実施の形態３）

次に本発明の第３の実施の形態であるプログラムリンクプログラム、プログラムリンク方法、プログラムリンク装置について説明する。但し、前記第２の実施の形態と本質的に異なる部分についてのみ説明し、前記第２の実施の形態と本質的に同一部分については説明を省略する。

【０１３７】

この第３の実施の形態でも、前記第２の実施の形態と同様に、予めライブラリを所定の順序に従って並べ、その順に増加量をリンク総和量に加えることによってそのライブラリがリンク可能か否かを判断することに関し、前記第２の実施の形態と同じである。

【０１３８】

そのための処理手順を図７に示す。

【０１３９】

この図７に示す本発明の第３の実施の形態の処理手順に関し、前記図６に示す

本発明の第２の実施の形態の処理手順と本質的に相違するのは、予め全ライブラリを増加量でソートし、この増加量が小さなものから順にインデックス番号を付すことのみである（図７、７２３）。

【０１４０】

この増加量とは、前記の通り、各ライブラリ毎にそのライブラリがアプリケーションからリンクされている数（共有数（ｎ））マイナス１とそのライブラリのサイズとの積である（図７、７２２）。

【０１４１】

これは前記の通り、そのライブラリをリンクすることによって新たに必要となるメモリの増加量に相当する。

【０１４２】

そしてインデックス番号順に１つずつライブラリをリンクすることによってメモリオーバーフローを生じることが無いか否かを判断することは、前記第２の実施の形態と同じであるが（図７、７２５～７２９）、この第３の実施の形態ではインデックス番号が増加量の小さな順に付されているため、この増加量の順にライブラリのリンクの可否を判断し、可能な限り増加量が小さなライブラリからリンクが実行されることについてのみ、前記第２の実施の形態と相違する。

【０１４３】

この第３の実施の形態が前記第２の実施の形態と本質的に異なる部分は、以上説明した部分だけであるので、他に関しては説明を省略する。

【０１４４】

（実施の形態４）

次に本発明の第４の実施の形態であるプログラムリンクプログラム、プログラムリンク方法、プログラムリンク装置について説明する。但し、前記第２の実施の形態と本質的に異なる部分についてのみ説明し、前記第２の実施の形態と本質的に同一部分については説明を省略する。

【０１４５】

この第４の実施の形態でも、前記第２の実施の形態と同様に、予めライブラリを所定の順序に従って並べ、その順に増加量をリンク総和量に加えることによっ

てそのライブラリがリンク可能か否かを判断することに関し、前記第２の実施の形態と同じである。

【０１４６】

そのための処理手順を図８に示す。

【０１４７】

この図８に示す本発明の第４の実施の形態の処理手順に関し、前記図６に示す本発明の第２の実施の形態の処理手順と本質的に相違するのは、予め全ライブラリについてそのライブラリがリンクされているアプリケーションを調べる（図８、８２１）のと同時に、そのアプリケーションが使われる頻度を調べ（図８、８２２）、全ライブラリをこの使用頻度でソートし、この使用頻度が大きなものから順にインデックス番号を付すことのみである（図８、８２３）。

【０１４８】

例えば、あるライブラリが電子メール閲覧アプリケーションソフトによってリンクされているならば、その電子メール閲覧アプリケーションソフトの使用頻度は比較的高い、例えば１日平均５回程度であるかも知れない。

【０１４９】

或いは、あるライブラリが銀行振込アプリケーションソフトによってリンクされているならば、その銀行振込アプリケーションソフトの使用頻度は比較的低い、例えば平均１年間に数回程度であるかも知れない。

【０１５０】

例えばこのようなケースでは、前記電子メール閲覧アプリケーションソフトによってリンクされているライブラリの方が、銀行振込アプリケーションソフトによってリンクされているライブラリよりも小さなインデックス番号が付され、優先的にリンクが実行される。

【０１５１】

又、例えば１つのライブラリが複数のアプリケーションによってリンクされていることもある。例えば、ある比較的汎用性の高いライブラリは前記電子メール閲覧アプリケーションソフトと銀行振込アプリケーションソフトの両方でリンクされているかも知れない。

【０１５２】

このようなライブラリに関しては、そのライブラリをリンクしているアプリケーションソフトの使用頻度をどのようにして求めるかに関しては種々の方法が考えられる。

【０１５３】

例えば、そのライブラリをリンクしているアプリケーションソフトの中で最も使用頻度の高いものを用いても良いし、平均値を用いても良いし、中央値を用いることもできるし、総和を用いても構わない。

【０１５４】

最も使用頻度の高いアプリケーションソフトの使用頻度或いは、そのライブラリをリンクしている全アプリケーションソフトの使用頻度の総和を用いることが適当かも知れないが、その方法に限定されるものではない。

【０１５５】

このようにして、ライブラリをリンクしているアプリケーションソフトの使用頻度を基準に、リンクを実行するライブラリを決定することによって、比較的使用頻度の高いアプリケーションソフトの実行速度を向上させることができ、より使い勝手を良くすることができる。

【０１５６】

インデックス番号順に１つずつライブラリをリンクすることによってメモリオーバーフローを生じることが無いか否かを判断することは、前記第２の実施の形態と同じである（図８、８２５～８２９）。

【０１５７】

この第４の実施の形態が前記第２の実施の形態と本質的に異なる部分は、以上説明した部分だけであるので、他に関しては説明を省略する。

【０１５８】

（実施の形態５）

次に本発明の第５の実施の形態であるプログラムリンクプログラム、プログラムリンク方法、プログラムリンク装置について説明する。但し、前記第２の実施の形態と本質的に異なる部分についてのみ説明し、前記第２の実施の形態と本質

的に同一部分については説明を省略する。

【０１５９】

この第５の実施の形態でも、前記第２の実施の形態と同様に、予めライブラリを所定の順序に従って並べ、その順に増加量をリンク総和量に加えることによってそのライブラリがリンク可能か否かを判断することに関し、前記第２の実施の形態と同じである。

【０１６０】

この第５の実施の形態の処理手順に関し、本発明の第２の実施の形態の処理手順と本質的に相違するのは、予め全ライブラリをそのサイズによってソートし、そのサイズの小さな順にインデックス番号を付し、このインデックス番号順にリンクが可能か否かを判断することについてのみである。

【０１６１】

このようにすることによって、比較的メモリサイズの小さなライブラリを優先的にリンクすることができる。

【０１６２】

この第５の実施の形態が前記第２の実施の形態と本質的に異なる部分は、以上説明した部分だけであるので、他に関しては説明を省略する。

【０１６３】

（実施の形態６）

次に本発明の第６の実施の形態であるプログラムリンクプログラム、プログラムリンク方法、プログラムリンク装置について説明する。但し、前記第２の実施の形態と本質的に異なる部分についてのみ説明し、前記第２の実施の形態と本質的に同一部分については説明を省略する。

【０１６４】

この第６の実施の形態でも、前記第２の実施の形態と同様に、予めライブラリを所定の順序に従って並べ、その順に増加量をリンク総和量に加えることによってそのライブラリがリンク可能か否かを判断することに関し、前記第２の実施の形態と同じである。

【０１６５】

この第６の実施の形態の処理手順に関し、本発明の第２の実施の形態の処理手順と本質的に相違するのは、予め全ライブラリをその実行時リンクに必要な時間によってソートし、その時間が大きなものから順にインデックス番号を付し、このインデックス番号順にリンクが可能か否かを判断することについてのみである。

【０１６６】

このようにすることによって、実行時リンクに必要な時間が大きなライブラリを優先的にリンクすることができ、全体としての実行速度を改善することができる。

【０１６７】

この第６の実施の形態が前記第２の実施の形態と本質的に異なる部分は、以上説明した部分だけであるので、他に関しては説明を省略する。

【０１６８】

【発明の効果】

以上詳細に説明したように、本発明によるならば、端末装置のメモリ容量を最大限に使用しながら、また同時にアプリケーション実行時の速度を最大限向上させることができ、その実用的効果は極めて大きい。

【０１６９】

またこの効果は特にメモリ容量に制限の大きな携帯機器では更に大きなものとなる。

【図面の簡単な説明】

【図１】

本発明の第１の実施の形態のリンクを実行する端末装置のブロック構成図

【図２】

本発明の第１の実施の形態のリンクを実行する端末装置の処理流れ図

【図３】

本発明の第１の実施の形態のリンクの変形を実行する端末装置の処理流れ図

【図４】

本発明の第１の実施の形態のリンクが実行される過程のクラスライブラリの構

造変化を示す図

【図５】

本発明の第１の実施の形態のリンクが実行される過程のクラスライブラリ数とリンク数を示す図

【図６】

本発明の第２の実施の形態のリンクを実行する端末装置の処理流れ図

【図７】

本発明の第３の実施の形態のリンクを実行する端末装置の処理流れ図

【図８】

本発明の第４の実施の形態のリンクを実行する端末装置の処理流れ図

【図９】

従来のＪａｖａ（Ｒ）リンカの実行例（実行時リンク方式）を示す図

【図１０】

従来のＣリンカの実行例（事前リンク方式）を示す図

【符号の説明】

１０１，９０１，９２１　端末装置

１０９，９０９，９２９　サーバ

１１３　ダウンロード命令

１１５　事前リンク命令

１１７　実行命令

１３１　制御部

１３３　ダウンロード制御部

１３５　結合ファイル解除部

１３７　ファイルシステム部

１４１　事前リンク部

１４２　リンク数検出部

１４３　結合部

１４４　メモリ量管理部

１４６　クラス呼び出し関係検出部

１５１　実行制御部

１５２　実行時リンク部

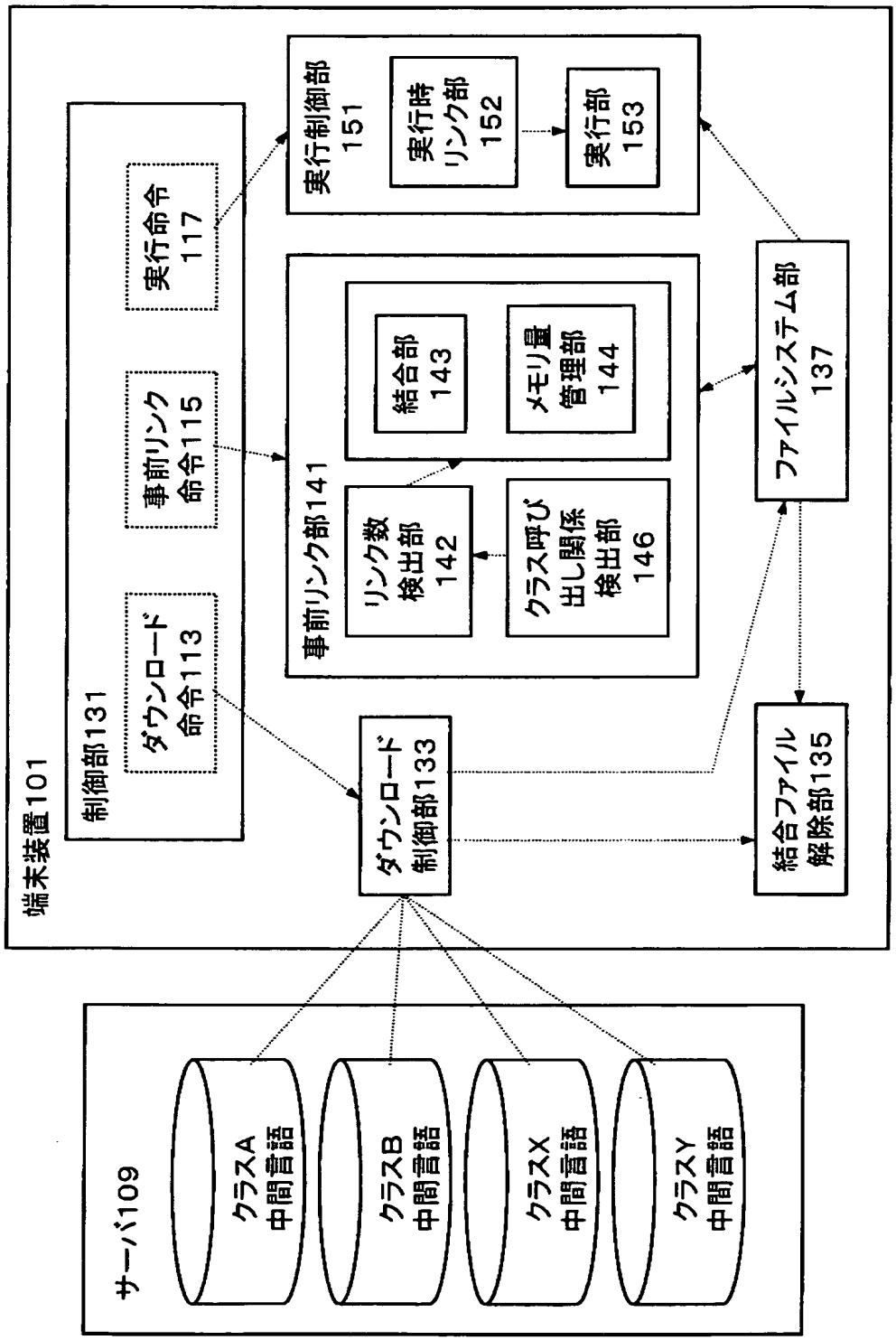
１５３　実行部

９０２，９０３，９２２，９２３　アプリケーションプログラム

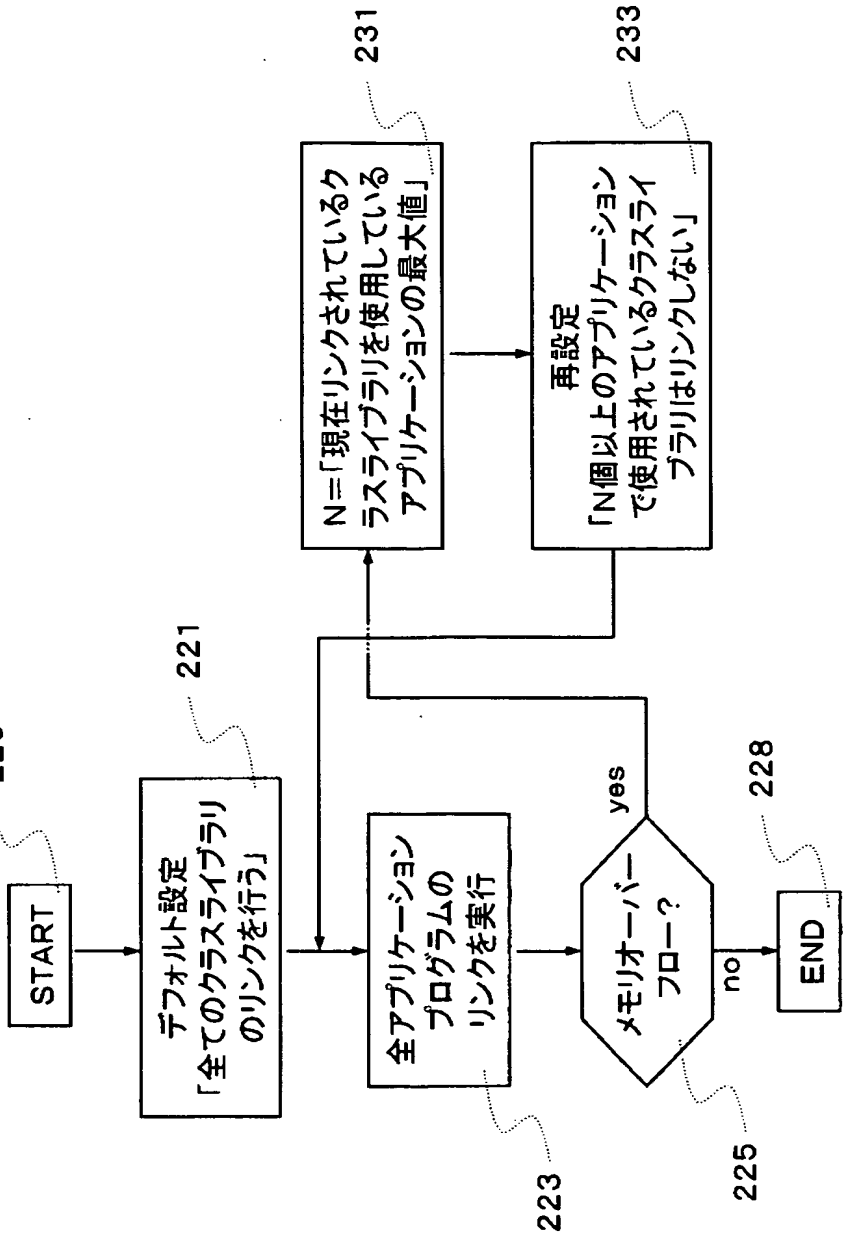
【書類名】　　図面

【図１】

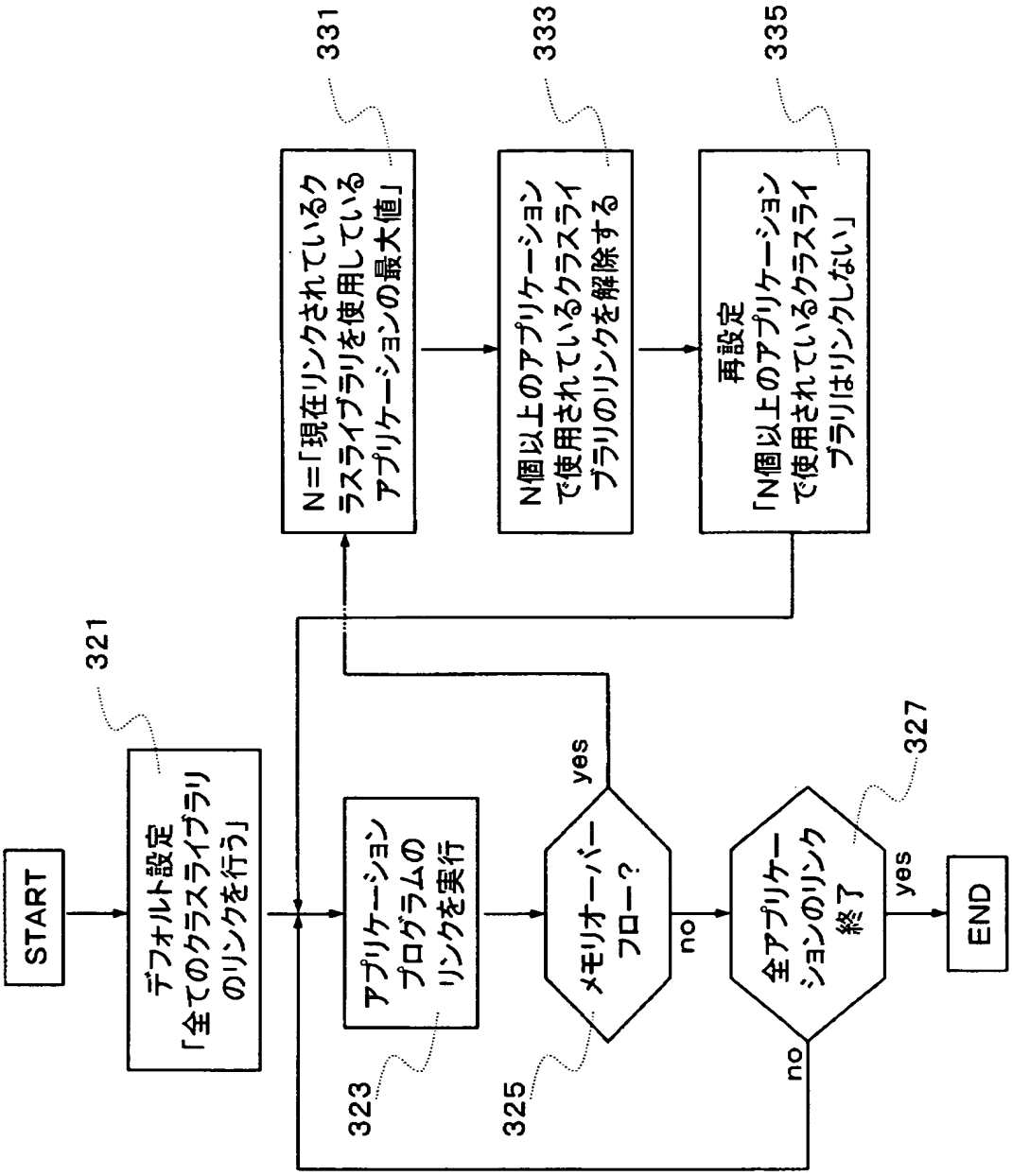
本発明の第1の実施の形態のリンクを実行する端末装置のブロック構成図



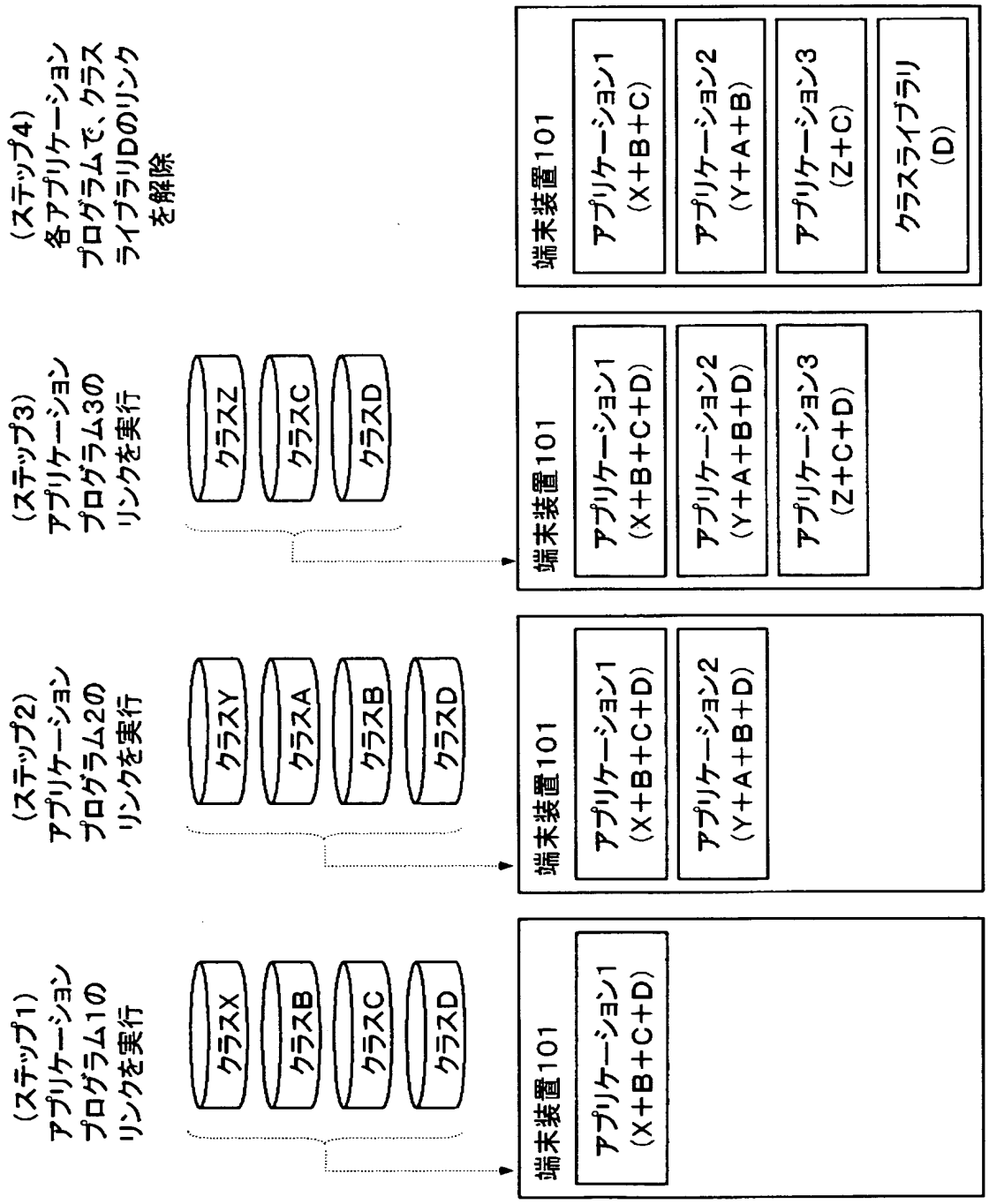
【図２】


220
START
221
デフォルト設定
「全てのクラスライブラリ
のリンクを行う」
223
全アプリケーション
プログラムの
リンクを実行
225
メモリオーバー
フロー？
yes
no
228
END
231
N=「現在リンクされているク
ラスライブラリを使用してい
るアプリケーションの最大値」
233
再設定
「N個以上のアプリケーション
で使用されているクラスライ
ブラリはリンクしない」

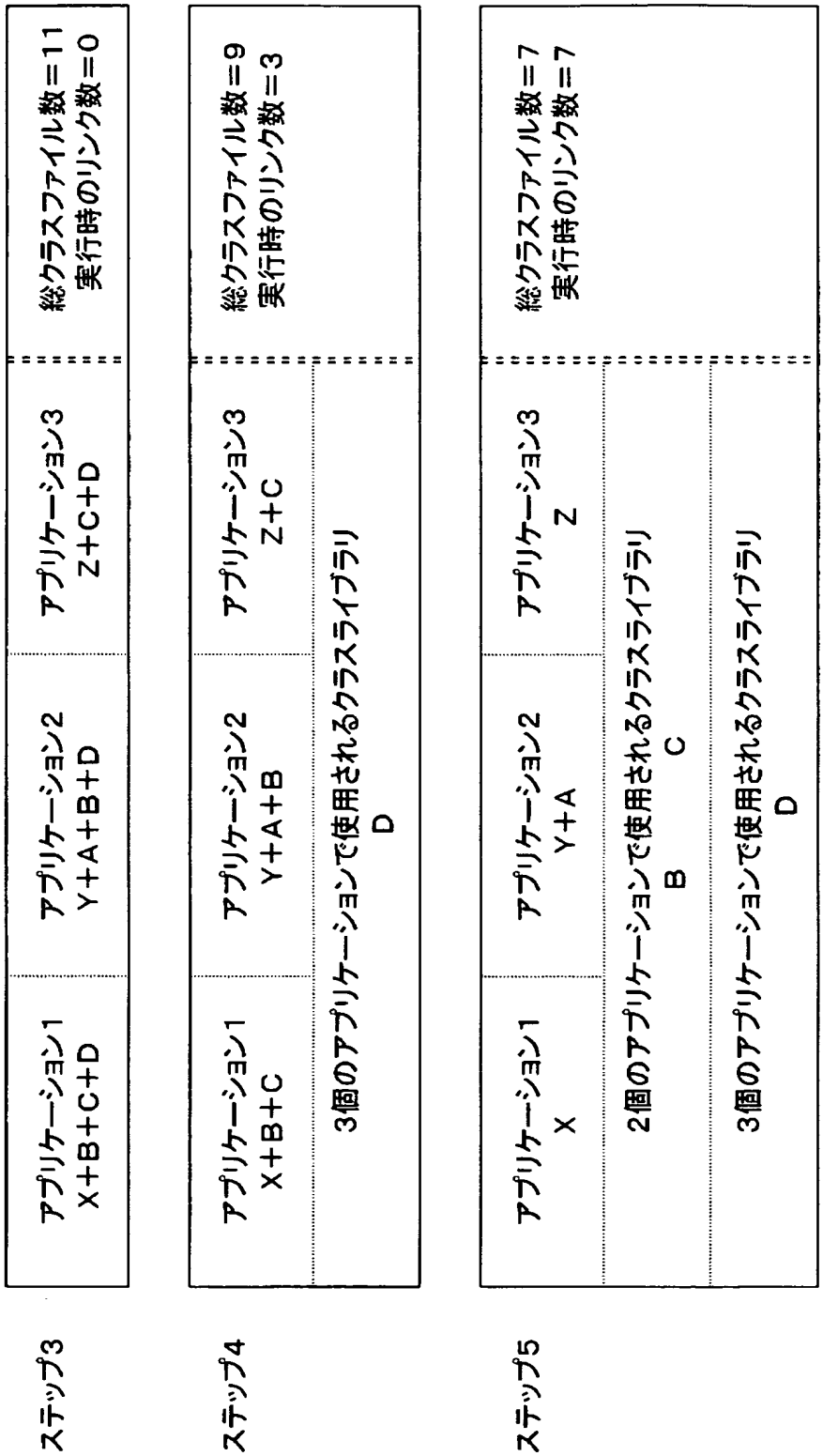
【図３】


START
デフォルト設定
「全てのクラスライブラリ
のリンクを行う」
321
アプリケーション
プログラムの
リンクを実行
323
メモリオーバー
フロー？
325
yes
no
全アプリケー
ションのリンク
終了
327
yes
no
END
N=「現在リンクされているク
ラスライブラリを使用してい
るアプリケーションの最大値」
331
N個以上のアプリケーション
で使用されているクラスライ
ブラリのリンクを解除する
333
再設定
「N個以上のアプリケーション
で使用されているクラスライ
ブラリはリンクしない」
335

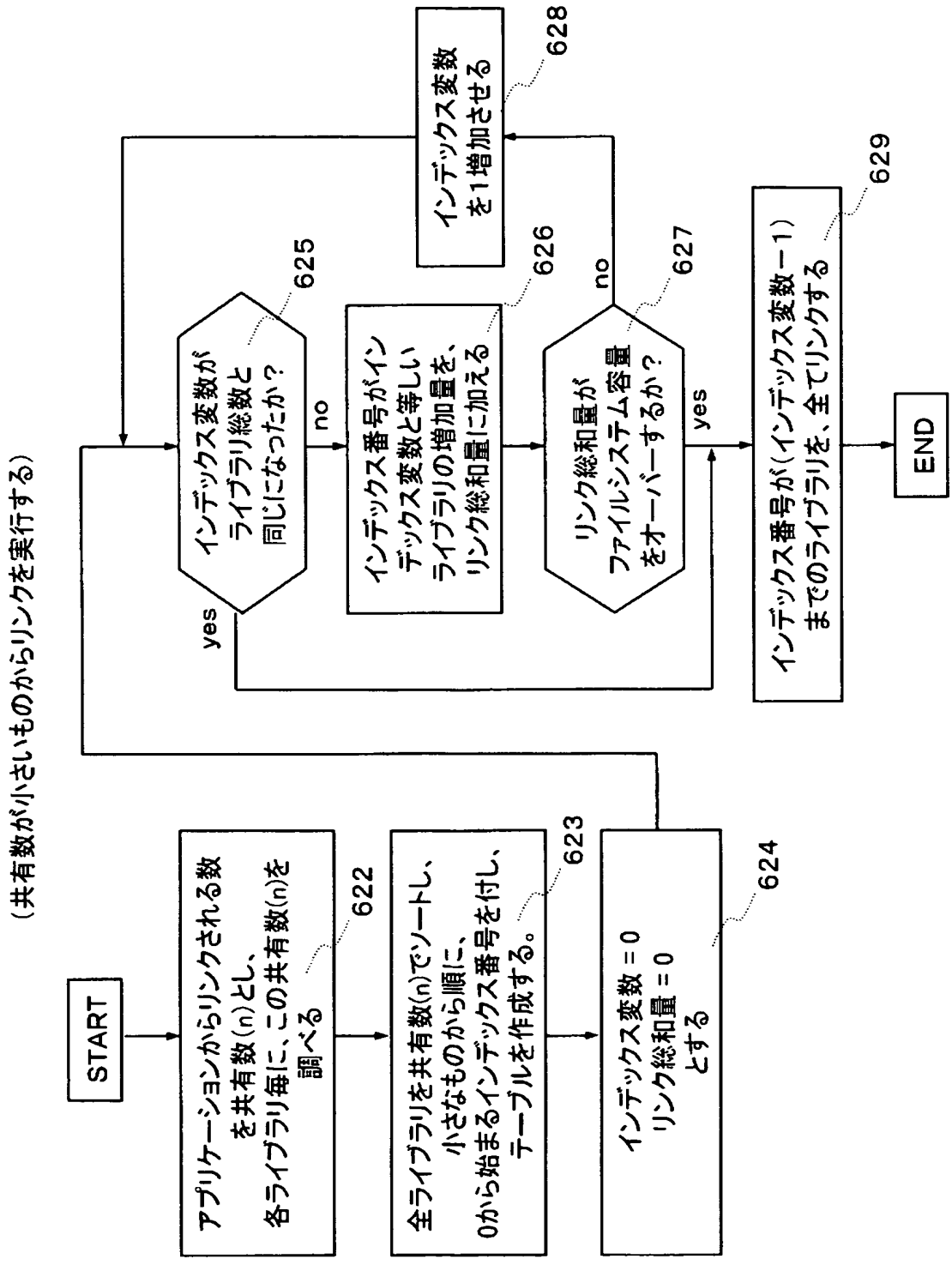
【図４】


(ステップ1)
アプリケーション
プログラム1の
リンクを実行
クラスX
クラスB
クラスC
クラスD
端末装置101
アプリケーション1
(X+B+C+D)
(ステップ2)
アプリケーション
プログラム2の
リンクを実行
クラスY
クラスA
クラスB
クラスD
端末装置101
アプリケーション1
(X+B+C+D)
アプリケーション2
(Y+A+B+D)
(ステップ3)
アプリケーション
プログラム3の
リンクを実行
クラスZ
クラスC
クラスD
端末装置101
アプリケーション1
(X+B+C+D)
アプリケーション2
(Y+A+B+D)
アプリケーション3
(Z+C+D)
(ステップ4)
各アプリケーション
プログラムで、クラス
ライブラリDのリンク
を解除
端末装置101
アプリケーション1
(X+B+C)
アプリケーション2
(Y+A+B)
アプリケーション3
(Z+C)
クラスライブラリ
(D)

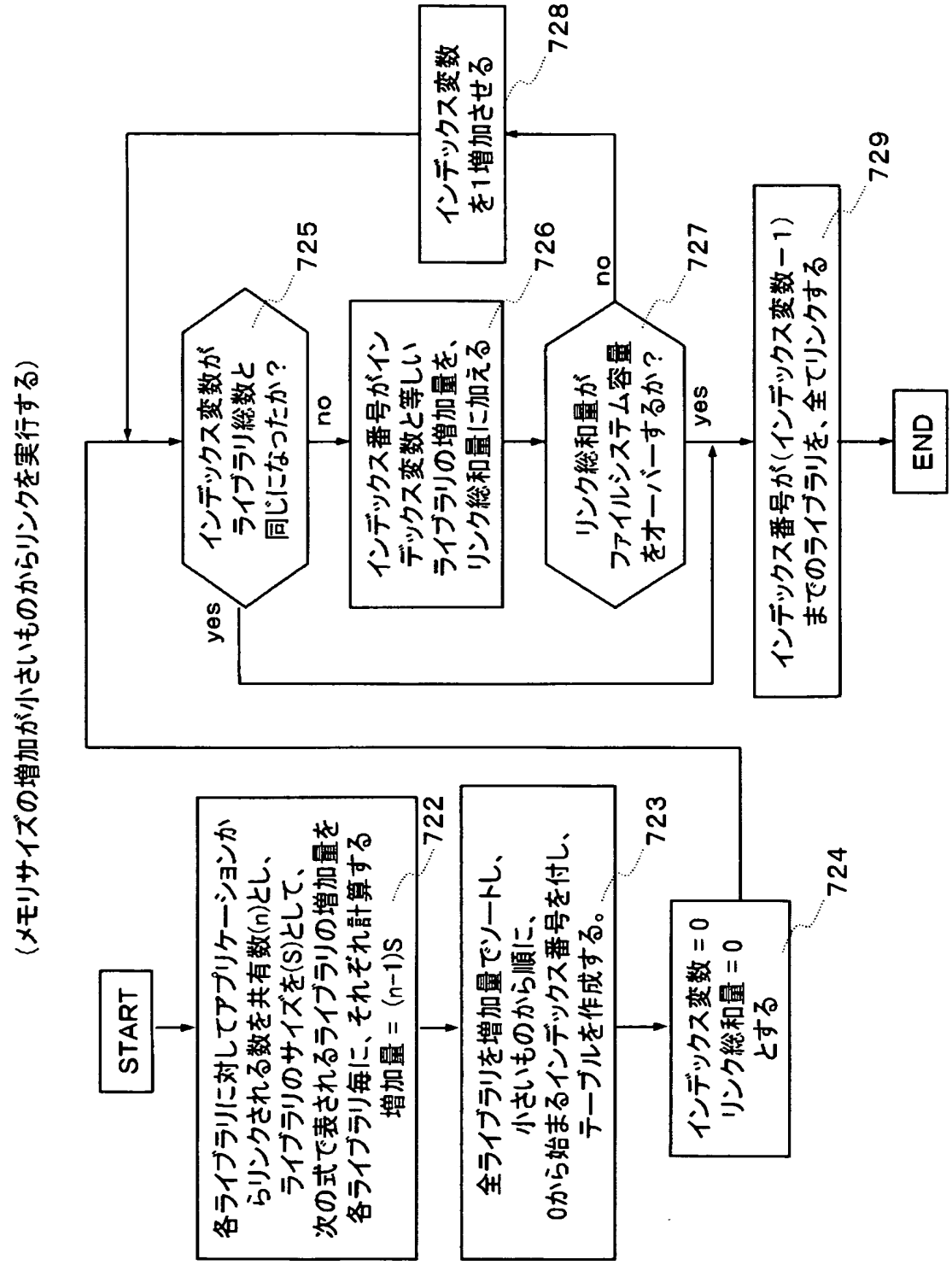
【図５】


ステップ3
アプリケーション1
X+B+C+D
アプリケーション2
Y+A+B+D
アプリケーション3
Z+C+D
総クラスファイル数＝11
実行時のリンク数＝0
ステップ4
アプリケーション1
X+B+C
アプリケーション2
Y+A+B
アプリケーション3
Z+C
総クラスファイル数＝9
実行時のリンク数＝3
3個のアプリケーションで使用されるクラスライブラリ
D
ステップ5
アプリケーション1
X
アプリケーション2
Y+A
アプリケーション3
Z
総クラスファイル数＝7
実行時のリンク数＝7
2個のアプリケーションで使用されるクラスライブラリ
B　　C
3個のアプリケーションで使用されるクラスライブラリ
D

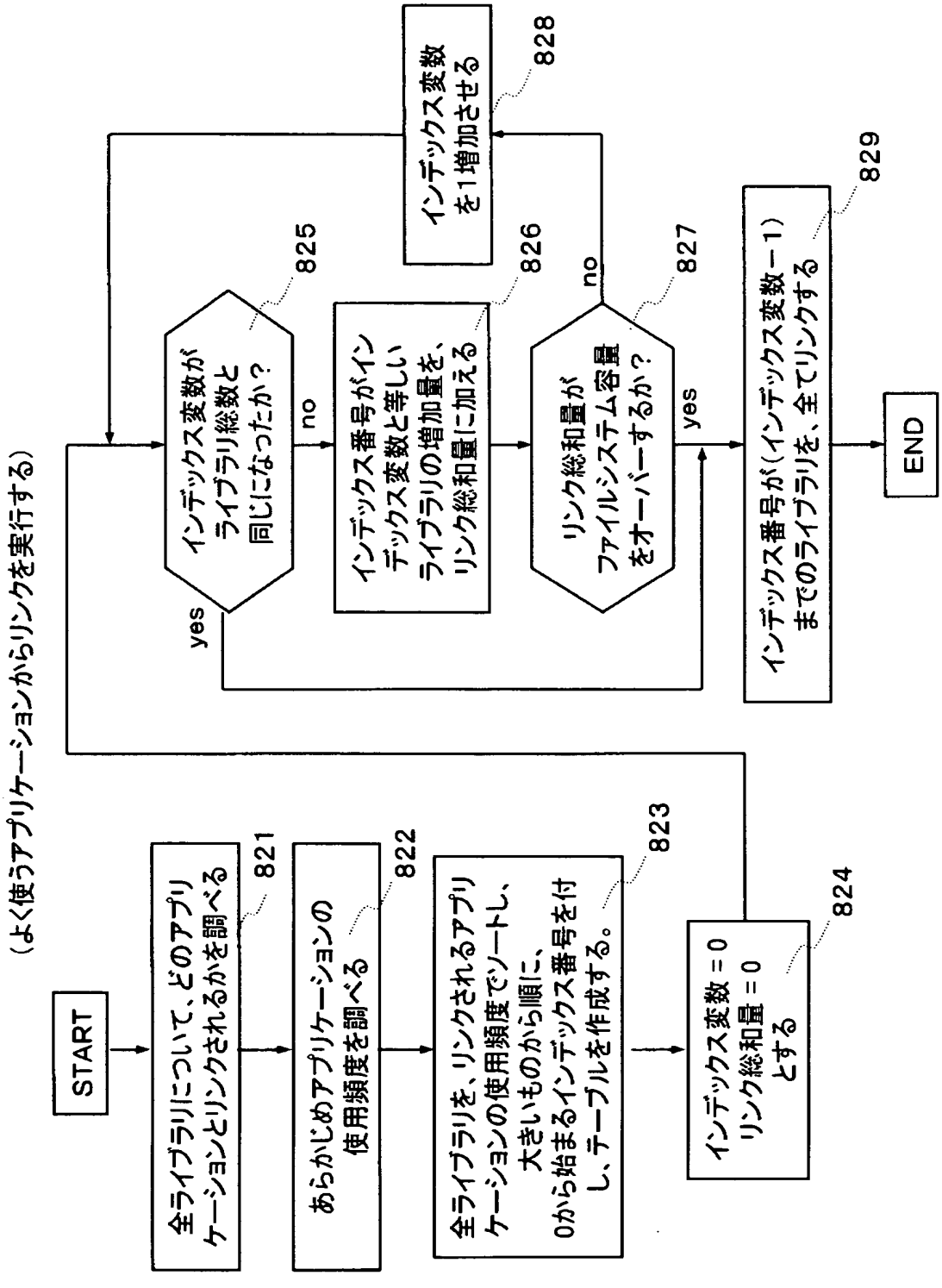
【図６】


(共有数が小さいものからリンクを実行する)
START
アプリケーションからリンクされる数を共有数(n)とし、各ライブラリ毎に、この共有数(n)を調べる
622
全ライブラリを共有数(n)でソートし、小さなものから順に、0から始まるインデックス番号を付し、テーブルを作成する。
623
インデックス変数 = 0
リンク総和量 = 0
とする
624
インデックス変数がライブラリ総数と同じになったか？
625
yes
no
インデックス番号がインデックス変数と等しいライブラリの増加量を、リンク総和量に加える
626
リンク総和量がファイルシステム容量をオーバーするか？
627
yes
no
インデックス変数を1増加させる
628
インデックス番号が(インデックス変数－1)までのライブラリを、全てリンクする
629
END

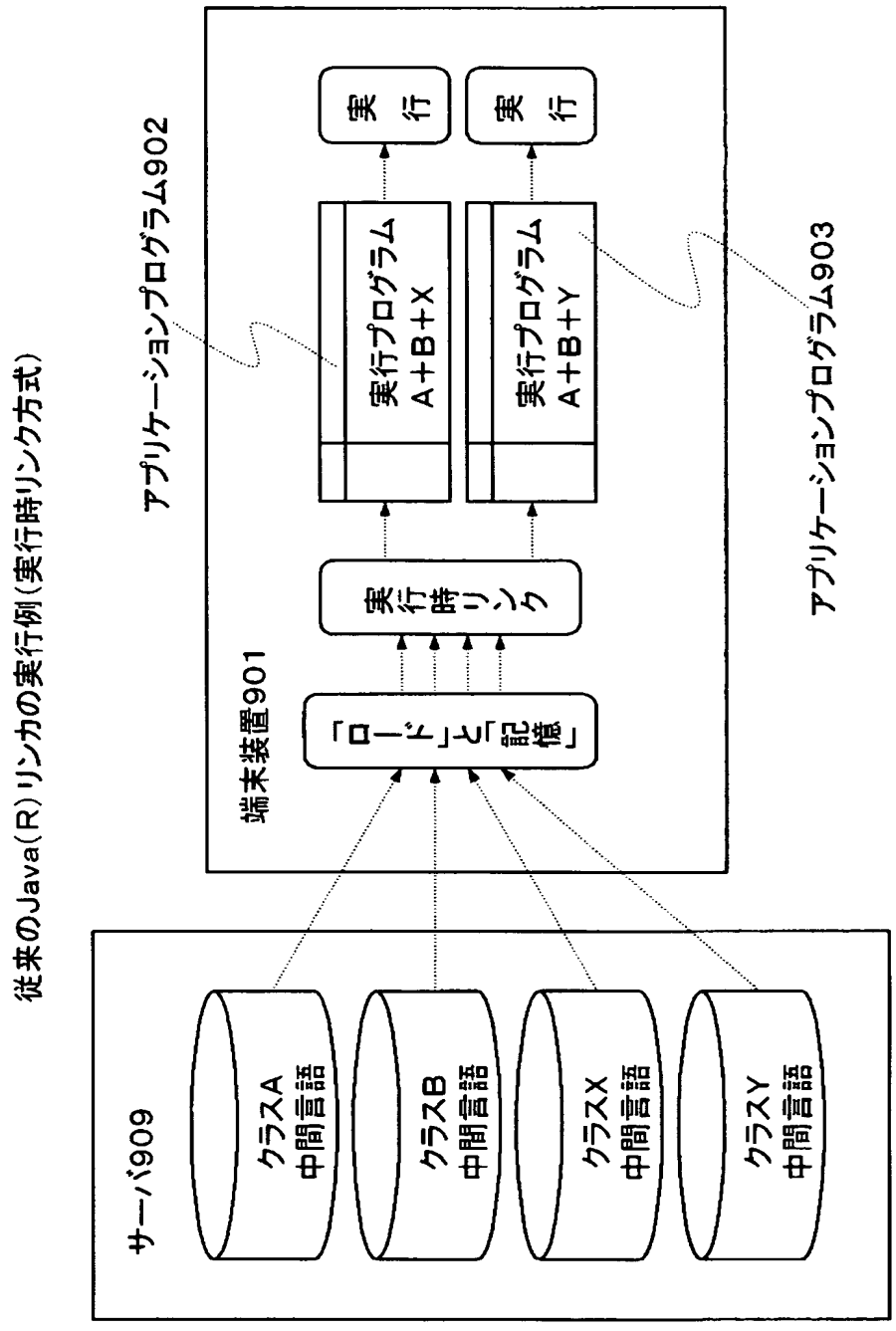
【図７】


（メモリサイズの増加が小さいものからリンクを実行する）
START
各ライブラリに対してアプリケーションからリンクされる数を共有数(n)とし、ライブラリのサイズを(S)として、次の式で表されるライブラリの増加量を各ライブラリ毎に、それぞれ計算する
増加量 = (n-1)S
722
全ライブラリを増加量でソートし、小さいものから順に、0から始まるインデックス番号を付し、テーブルを作成する。
723
インデックス変数 = 0
リンク総和量 = 0
とする
724
インデックス変数がライブラリ総数と同じになったか？
725
yes
no
インデックス番号がインデックス変数と等しいライブラリの増加量を、リンク総和量に加える
726
リンク総和量がファイルシステム容量をオーバーするか？
727
no
yes
インデックス変数を1増加させる
728
インデックス番号が(インデックス変数－1)までのライブラリを、全てリンクする
729
END

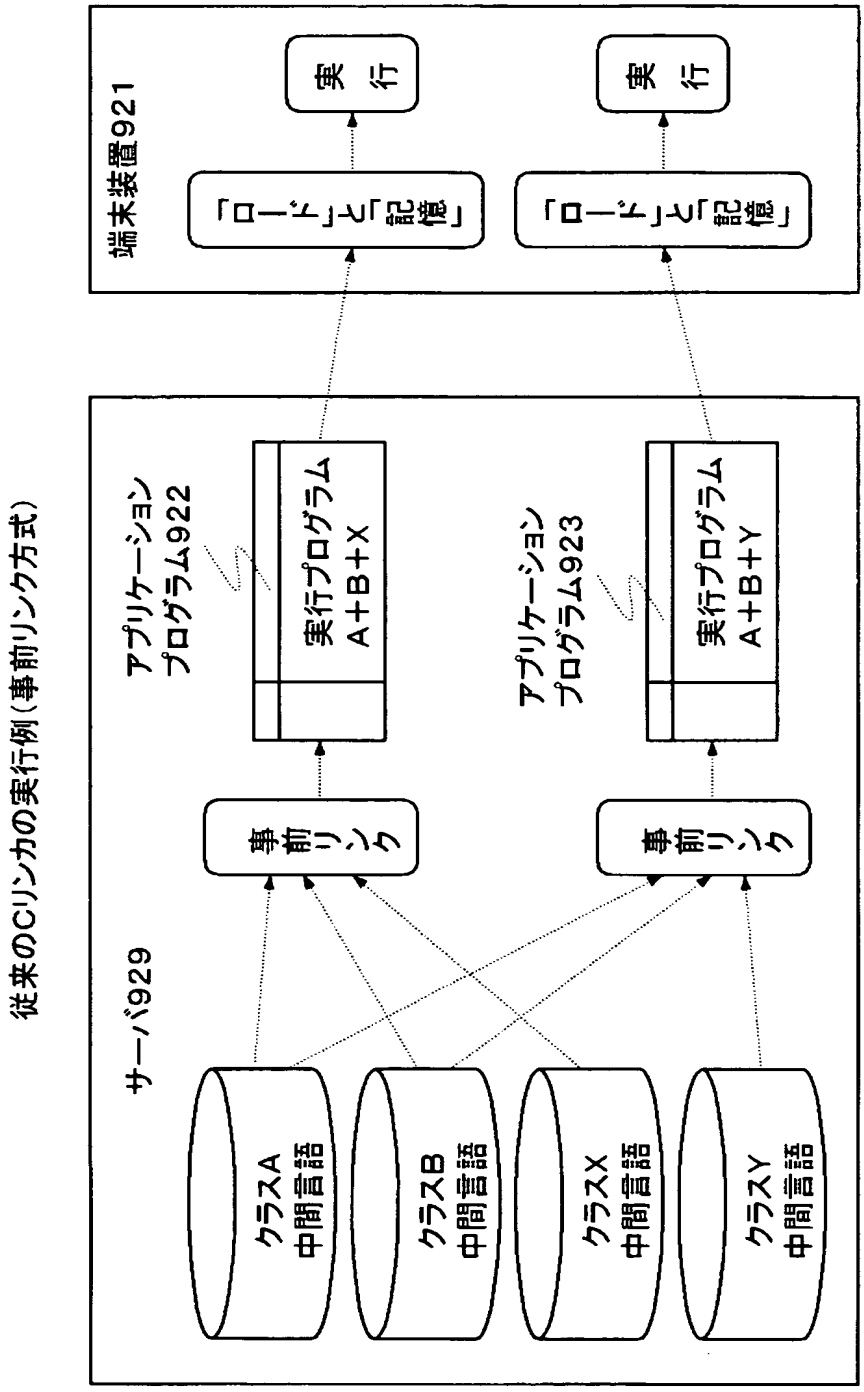
【図８】


(よく使うアプリケーションからリンクを実行する)
START
全ライブラリについて、どのアプリケーションとリンクされるかを調べる
821
あらかじめアプリケーションの使用頻度を調べる
822
全ライブラリを、リンクされるアプリケーションの使用頻度でソートし、大きいものから順に、0から始まるインデックス番号を付し、テーブルを作成する。
823
インデックス変数 = 0
リンク総和量 = 0
とする
824
インデックス変数がライブラリ総数と同じになったか？
825
yes
no
インデックス番号がインデックス変数と等しいライブラリの増加量を、リンク総和量に加える
826
リンク総和量がファイルシステム容量をオーバーするか？
827
yes
no
インデックス変数を1増加させる
828
インデックス番号が(インデックス変数－1)までのライブラリを、全てリンクする
829
END

【図９】


従来のJava(R)リンカの実行例(実行時リンク方式)
サーバ909
クラスA 中間言語
クラスB 中間言語
クラスX 中間言語
クラスY 中間言語
端末装置901
「ロード」と「記憶」
実行時リンク
実行プログラム A+B+X
実行プログラム A+B+Y
実行
実行
アプリケーションプログラム902
アプリケーションプログラム903

【図１０】


従来のCリンカの実行例(事前リンク方式)
端末装置921
実　行
実　行
「ロード」と「記憶」
「ロード」と「記憶」
アプリケーション
プログラム922
アプリケーション
プログラム923
実行プログラム
A+B+X
実行プログラム
A+B+Y
事前リンク
事前リンク
サーバ929
クラスA
中間言語
クラスB
中間言語
クラスX
中間言語
クラスY
中間言語

【書類名】　要約書

【要約】

【課題】　限られたメモリ容量で、可能な限りライブラリを予めリンクしておくことにより、アプリケーション実行時の起動速度を改善する。

【解決手段】　複数のリンク前プログラムから１または複数個をリンクしてリンク後プログラムを作成し、前記リンク後プログラムを記憶手段に記憶し、前記記憶手段の使用状況を判断し、前記記憶手段の使用状況の判断に応じて前記リンク後プログラムを作成するために必要なリンクの一部だけを行うかあるいは全く行わないプログラムリンクプログラム。

【選択図】　図１

特願２００３－０９７８９２

出　願　人　履　歴　情　報

識別番号　　［０００００５８２１］

１．変更年月日　　１９９０年　８月２８日
［変更理由］　　新規登録
住　所　　大阪府門真市大字門真１００６番地
氏　名　　松下電器産業株式会社